

# **Perspektiverende Datalogi** *Internetalgoritmer*


## **MapReduce**

Gerth Stølting Brodal

# MapReduce Implementationen

Dean, F. and Ghemawat, S. (2004) *MapReduce: Simplified Data Processing on Large Clusters*.  
In: Sixth Symposium on Operating System Design and Implementation (OSDI 2004): 137-150

MapReduce

The Google logo, featuring the word "Google" in its characteristic multi-colored font (blue, red, yellow, blue, green, red) with a trademark symbol.

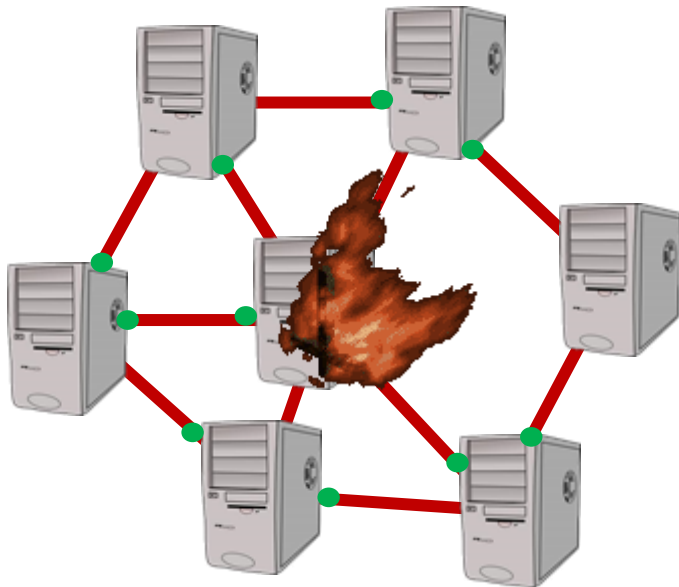
Hadoop

Apache open  
source projekt

The Yahoo! logo, featuring the word "YAHOO!" in a red, serif font with a registered trademark symbol.The Facebook logo, consisting of the word "facebook" in white, lowercase letters on a blue rectangular background.The Amazon.com logo, featuring the word "amazon.com" in black, lowercase letters with a curved orange arrow underneath.

# Parallele algoritmer – Teori vs Praksis

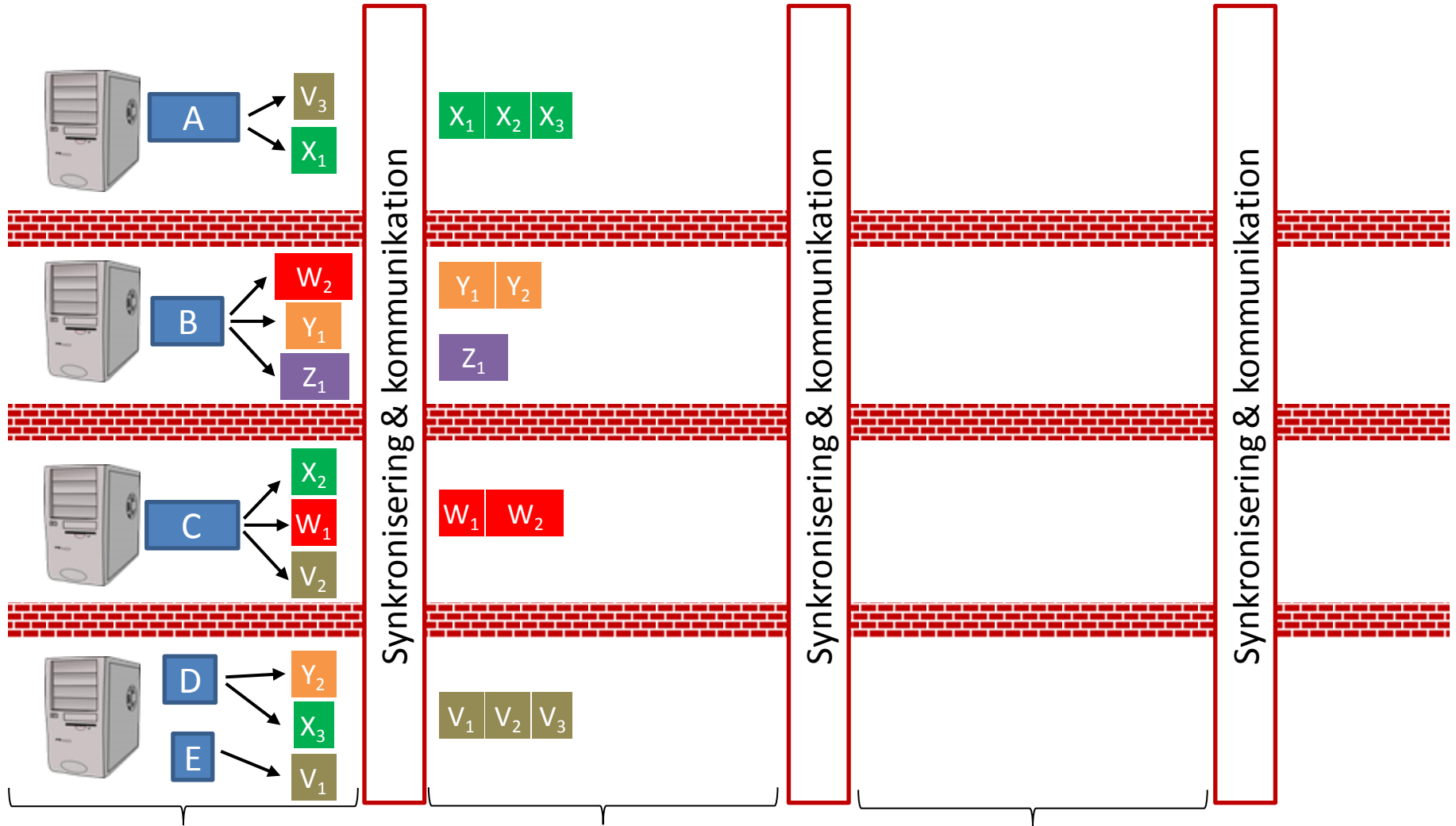
- Behandling af massiv data kræver parallelle algoritmer (fx Google)
- **Ideelle Teoretiske Verden**
  - mange maskiner samarbejder synkront
  - kan kommunikere i hvert beregningskridt
  - utallige teoretiske algoritmer udviklet



- **Praksis**
  - meget svært at programmere
  - korrekthed, concurrency problemer, ...
  - maskiner arbejder med forskellige hastigheder, strejker, bryder sammen

**Ønskes:** Simpel men slagkraftig parallel model


# Afgrænsning af Kommunikation



Beregning på uafhængige maskiner

→ tid

# Parallele Programmer

- Traditionelt specielt udviklede programmer for hvert problem man ønsker at løse
  - mange ikke-trivielle detaljer i parallel programmer
  - fejl-tolerance (1000'er af maskiner fejler regelmæssigt)
  - fordeling af opgaver blandt maskiner
  - balancering af arbejdet blandt maskiner
- **MapReduce** interfacet (, 2003)
  - håndter ovenstående automatisk
  - meget begrænsede kommunikation mellem maskiner
  - algoritmen udføres i **Map** og **Reduce** faser

# MapReduce

- Brugeren skal definere to funktioner

**map**            v1                            → List[ (k,v2) ]

**reduce**        (k, List[ v2 ]) → List[ v3 ]

- **Eksempel:** Antal forekomster af ord i en tekstsamling

**map** { ("www.foo.com", "der var en gang en...") → [ ("der", "1"), ("var", "1"), ("en", "1"), ("gang", "1"), ("en", "1")... ]  
      ( "www.bar.com", "en lang gang..." ) → [ ("en", "1"), ("lang", "1"), ("gang", "1"), ... ]

**reduce** { ("en", [ "1", "1", "1" ])                            → [ "en 3" ]  
          ( "gang", [ "1", "1" ] )                            → [ "gang 2" ]  
          ...

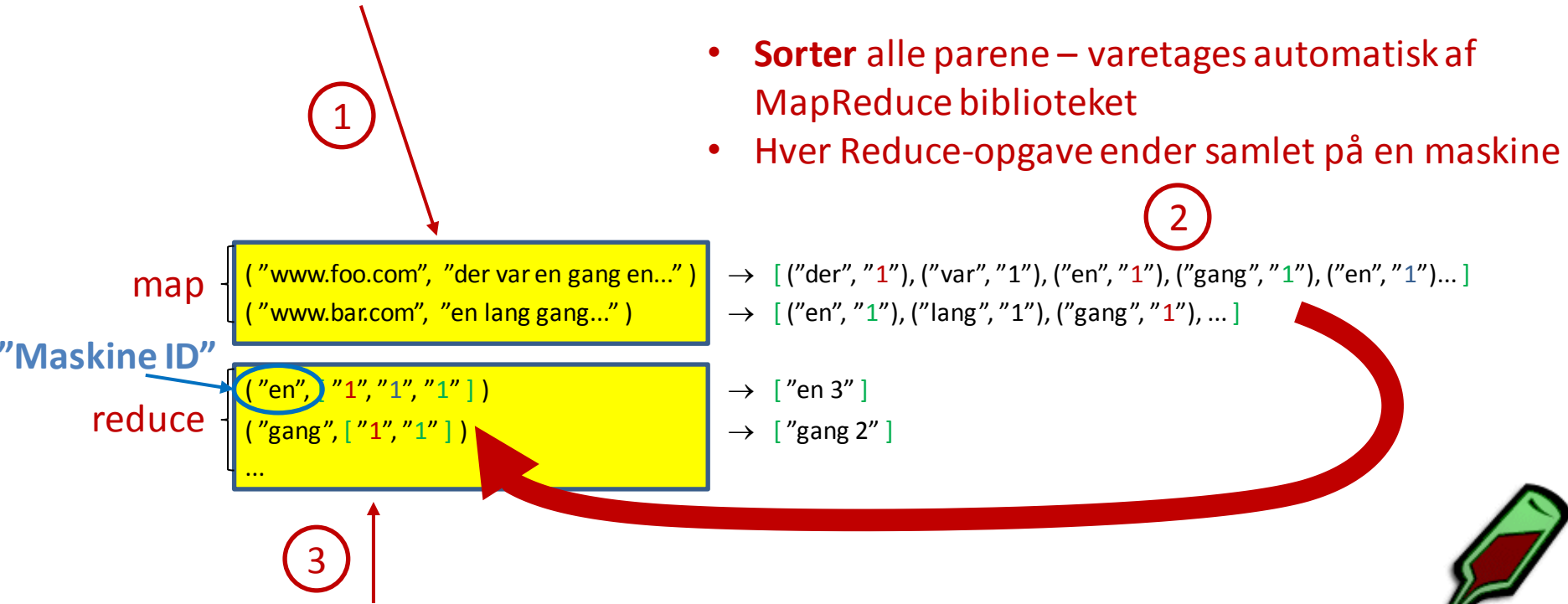
- Output fra en map-reduce kan være input til den næste map-reduce

# MapReduce

- Map-opgaverne ligger spredt ud på maskinerne (i et GFS hvor data typisk er replikeret 3 gange)
- Hver maskine udfører et antal uafhængige map-opgaver

**Master** = en maskine der

1. planlægger og fordeler opgaverne
2. genstarter hængene opgaver på andre maskiner



- **Sorter** alle parene – varetages automatisk af MapReduce biblioteket
- Hver Reduce-opgave ender samlet på en maskine

- En reduce-opgave løses sekventielt på én maskine (mulig FLASKEHALS )
- Hver maskine udfører et antal uafhængige reduce-opgaver
- Output er en del af den samlede output

**Antagelse** Antal maskiner  $\leq n^{1-\epsilon}$ , hver maskine hukommelse  $\leq n^{1-\epsilon}$

# Hadoop : WordCount

Java kode fra tutorialen på [hadoop.apache.org](http://hadoop.apache.org)

```
public class WordCount {
    public static class Map extends MapReduceBase
    implements Mapper<LongWritable, Text, Text, IntWritable> {
        private final static IntWritable one = new IntWritable(1);
        private Text word = new Text();
        public void map(LongWritable key, Text value, OutputCollector<Text, IntWritable> output,
            Reporter reporter) throws IOException {
            String line = value.toString();
            StringTokenizer tokenizer = new StringTokenizer(line);
            while (tokenizer.hasMoreTokens()) {
                word.set(tokenizer.nextToken());
                output.collect(word, one);
            }
        }
    }

    public static class Reduce extends MapReduceBase
    implements Reducer<Text, IntWritable, Text, IntWritable> {
        public void reduce(Text key, Iterator<IntWritable> values,
            OutputCollector<Text, IntWritable> output, Reporter reporter) throws IOException {
            int sum = 0;
            while (values.hasNext()) {
                sum += values.next().get();
            }
            output.collect(key, new IntWritable(sum));
        }
    }
    ...
}
```



# En søgemaskines dele

## Indsamling af data

- **Webcrawling** (gennemløb af internet)

## Indeksering data

- **Parsing** af dokumenter
- **Leksikon**: indeks (ordbog) over alle ord mødt
- **Inverteret fil**: for alle ord i leksikon, angiv i hvilke dokumenter de findes

## Søgning i data

- Find alle dokumenter med søgeordene
- **Rank** dokumenterne

# Inverteret fil

Input: `List[ (URL, tekst) ]`

Output: `List[ (Ord, URL'er) ]`

## Map

$(\text{URL}, \text{tekst}) \rightarrow [ (\text{ord}_1, \text{URL}), \dots, (\text{ord}_k, \text{URL}) ]$

## Reduce

$(\text{ord}, [ \text{URL}_1, \dots, \text{URL}_m ] ) \rightarrow [ (\text{ord}, \text{"URL}_1, \dots, \text{URL}_m\text{"}) ]$

# Indgrad af alle siderne

Input:  $\text{List}[ (i, j) ]$

Output:  $\text{List}[ (i, \text{indgrad}(i)) ]$

## Map

$$(i, j) \rightarrow [ (j, 1) ]$$

## Reduce

$$(i, \underbrace{[ 1, 1, \dots, 1 ]}_{k = \text{indgrad}(i)}) \rightarrow [ (i, k) ]$$

# Sum

Input:  $[x_1, \dots, x_n]$

Output:  $[\text{sum}(x_1, \dots, x_n)]$

## Map

$x_i \rightarrow [\text{random}(1..R), x_i]$

1

## Reduce

$(r, [x_{i_1}, \dots, x_{i_k}]) \rightarrow [x_{i_1} + \dots + x_{i_k}]$

$R \approx$  antal maskiner

## Map

$x_i \rightarrow [(0, x_i)]$

2

## Reduce

$(0, [x_1, \dots, x_n]) \rightarrow [x_1 + \dots + x_n]$

# Øvelser - PageRank

Input: `List[ (i, j) ]`

Output: `List[ (i,  $p_i^{(s)}$ ) ]`

$$p_i^{(s)} = 0.85 \cdot \sum_{j:j \rightarrow i} \frac{p_j^{(s-1)}}{\text{udgrad}(j)} + 0.15 \cdot \frac{1}{n}$$

$$p_1^{(0)} = 1.0 \quad p_2^{(0)} = \dots = p_n^{(0)} = 0.0$$

# PageRank

$$p_1^{(0)} = 1.0 \quad p_2^{(0)} = \dots = p_n^{(0)} = 0.0 \quad p_i^{(s)} = 0.85 \cdot \sum_{j:j \rightarrow i} \frac{p_j^{(s-1)}}{\text{udgrad}(j)} + 0.15 \cdot \frac{1}{n}$$

$$[(i_1, j_1), (i_2, j_2), \dots] \quad \textcircled{1} \rightarrow [(i_1, j_1, p_{i_1}^{(0)}), (i_2, j_2, p_{i_2}^{(0)}), \dots]$$

$$\textcircled{2} \rightarrow [(i_1, j_1, p_{i_1}^{(0)}, \text{udgrad}(i_1)), (i_2, j_2, p_{i_2}^{(0)}, \text{udgrad}(i_2)), \dots]$$

$$\textcircled{3} \rightarrow [(i_1, j_1, p_{i_1}^{(0)}, \text{udgrad}(i_1), n), (i_2, j_2, p_{i_2}^{(0)}, \text{udgrad}(i_2), n), \dots]$$

$$s \left\{ \begin{array}{l} \textcircled{4} \rightarrow [(i_1, j_1, p_{i_1}^{(1)}, \text{udgrad}(i_1), n), (i_2, j_2, p_{i_2}^{(1)}, \text{udgrad}(i_2), n), \dots] \\ \textcircled{4} \rightarrow [(i_1, j_1, p_{i_1}^{(2)}, \text{udgrad}(i_1), n), (i_2, j_2, p_{i_2}^{(2)}, \text{udgrad}(i_2), n), \dots] \\ \dots \\ \textcircled{4} \rightarrow [(i_1, j_1, p_{i_1}^{(s)}, \text{udgrad}(i_1), n), (i_2, j_2, p_{i_2}^{(s)}, \text{udgrad}(i_2), n), \dots] \end{array} \right.$$

$$\textcircled{5} \rightarrow [(i_1, p_{i_1}^{(s)}), (i_2, p_{i_2}^{(s)}), \dots]$$

**Hint**  $\textcircled{1}$ ,  $\textcircled{2}$  og  $\textcircled{5}$  er nemmest